Foundations of Artificial Intelligence

M. Helmert
S. Eriksson
Spring Term 2022

University of Basel
Computer Science

Exercise Sheet 2

Due: March 6, 2022

Important: for submission, consult the rules at the end of the exercise. Non-adherence to the rules will lead to your submission not being corrected.

Exercise 2.1 (1+1 marks)

Characterize the following environments by describing if they are *static / dynamic*, *deterministic / non-deterministic / stochastic*, *fully / partially / not observable*, *discrete / continuous*, and *single-agent / multi-agent*. Explain your answers.

(a) Monopoly

(b) Mars rover

Exercise 2.2 (0.5+0.5 marks)

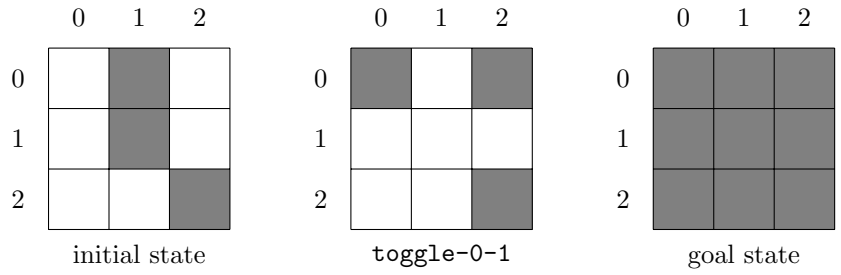
Determine if the following statements are true for all state spaces $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$. Explain your answers.

(a) Consider some state $s \in S$ and its predecessor $s' \in S$. All solutions π_s for s have lower or equal cost than all solutions π'_s for s'.

(b) If $cost(a) = cost(a') = x$ for all $a, a' \in A$, then we can define a state space $\mathcal{S}'$ where we replace all $a \in A$ with a fresh action a'. Formally, we define $\mathcal{S}' = \langle S, \{a'\}, \{a' \mapsto x\}, T', s_0, S_\star \rangle$ where $a' \notin A$ and $T' = \{\langle s, a', s' \rangle \mid \langle s, a, s' \rangle \in T\}$.

Exercise 2.3 (2 marks)

In the *lights-out* problem we have a grid of $n \times m$ cells, which can be *on* and *off*. Action `toggle-i-j` changes not only the status of the cell in row i and column j, but also of the cells above, below, left and right of it (if they exist). The action cost is equal to the number of cells that switch status, e.g., for a cell in a corner the action cost is 3, while for a cell in the center it is 5. The goal is to turn all the lights off.
The following figure shows an example instance where a white background represents a cell that is on, and a gray background a cell that is off:



Formalize the state space of the above lights-out problem instance. Specify all parts of the state space, i.e., the set of states, the set of actions, the cost function, the set of transitions, the initial state and the set of goal states.

Exercise 2.4 (3.5+1.5 marks)

The task in this exercise is to write a software program. We expect you to implement your code without using existing code you find online. If you encounter technical problems or have difficulties understanding the task, please let us know.

On the website of the course, you find Java code (`state-spaces.tar.gz`) that implements the black box interface for state spaces that was discussed in the lecture, as well as an interface for states and actions. The code includes an example implementation of the interface for the blocks world problem. You can test the implementation by invoking the `StateSpaceTest` class, which creates a set of random successor states starting from the initial state and afterwards performs a so-called random walk, i.e., it iteratively picks a successor at random and expands it.

To run the program, first compile it with

```
javac StateSpaceTest.java
```

from a shell (on Linux) and then run it with

```
java StateSpaceTest blocks blocks-problem.txt
```

The sole purpose of the provided blocks world implementation is to serve as an example that helps you for your own implementation.

Your task is to implement the provided interface for the *lights-out* problem. Everything should be implemented in a single file called `LightsOutStateSpace.java`. **In your handin, only submit your `LightsOutStateSpace.java` as a solution to this exercise.**

(a) Implement the state space of the lights-out problem (for a variable grid size).

(b) Implement the `buildFromCmdline` function for the lights-out problem such that it parses an input file where the first line consists of two integers (spearated by whitespace) denoting first the number of rows and second the number of columns, and the following rows describe the grid such that a cell that is on is denoted by *1*, and a cell that is off by *0*. The file `lights-out-problem.txt` from the archive encodes the instance from Exercise 2.3. Make sure that your code recognizes invalid inputs such as invalid numbers in the cell description or incorrect specification of the number of rows and columns.

You can test your implementation by uncommenting the two indicated lines in the method `createStateSpace` of the `StateSpaceTest` class and then executing the command

```
java StateSpaceTest lights-out lights-out-problem.txt
```

Notice that the command uses the new keyword `lights-out` (rather than `blocks` as in the example above). Also note that the problem is not expected to be solved by the provided random walk.

Submission rules:

- Exercise sheets must be submitted in groups of two students. Please submit a single copy of the exercises per group (only one member of the group does the submission).
- Create a single PDF file (ending .pdf) for all non-programming exercises. Use a file name that does not contain any spaces or special characters other than the underscore "_". If you want to submit handwritten solutions, include their scans in the single PDF. Make sure it is in a reasonable resolution so that it is readable, but ensure at the same time that the PDF size is not astronomically large. Put the names of all group members on top of the first page. Either use page numbers on all pages or put your names on each page. Make sure your PDF has size A4 (fits the page size if printed on A4).

- For programming exercises, only create those code textfiles required by the exercise. Put your names in a comment on top of each file. Make sure your code compiles and test it. Code that does not compile or which we cannot successfully execute will not be graded.
- For the submission: if the exercise sheet does not include programming exercises, simply upload the single PDF. If the exercise sheet includes programming exercises, upload a ZIP file (ending .zip, .tar.gz or .tgz; *not* .rar or anything else) containing the single PDF and the code textfile(s) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly.
- Do not upload several versions to ADAM, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.